Moving Target Techniques:

Cyber Resilience through Randomization, Diversity, and Dynamism

Hamed Okhravi and Howard Shrobe

Overview:

The static nature of computer systems makes them vulnerable to cyber attacks. Consider a situation where an attacker wants to compromise a remote system running a specific application. The attacker need only find one vulnerability in a local copy of that application. Since all copies of that application are identical and static, the attacker can leverage that vulnerability to exploit the application on a remote machine. Worse yet, the same vulnerability can be exploited to attack thousands or millions of other machines that run the same application. Also, since the internals of the system changes little over time, the same attack is likely to succeed for a long time. The situation is exacerbated by the fact that any reconnaissance information collected on the system by the attackers will also be valid for a long time. This creates an imbalance in favor of attacks.

A promising approach to cyber resilience that attempts to rebalance the cyber landscape is known as cyber moving target (MT) (or just moving target) techniques. Moving target techniques change the static nature of computer systems to increase both the difficulty and the cost (in effort, time, and resources) of mounting attacks. Simply put, these techniques turn systems into moving targets that will be hard for cyber attackers to compromise. MT techniques leverage randomization, diversity, and dynamism to achieve resilience. Randomization refers to introducing non-determinism to the internal structures of a system while preserving its correct functionality; diversity refers to introducing heterogeneity among computer systems so that they cannot be compromised by the same attack; and dynamism refers to changing the properties of a system over time so that the same attack cannot compromise it in the future. MT technique can implement any subset of these three goals.

In order to understand the different domains of MT techniques, we focus on the component that is actually subject to movement. For ease of design and implementation, a computer often consists of multiple layers of software and hardware, commonly referred to as the software stack although the stack includes the hardware elements as well. Each layer relies on other layers for its proper operation and function.

Figure 1 presents one representation of such a layered design. At the very bottom of the software stack is the hardware components of the machine. These include the processor, the motherboard, the memory cards, and other peripheral devices and cards such as the sound card, video card, etc. Above this layer resides the operating system which is responsible for controlling and managing the hardware components and providing an abstraction of them to the application. The abstraction provided by the operating system is the key in interoperability and compatibility of the applications because the applications do not typically interact directly with the hardware components; rather, they use the provided operating system abstraction. The abstraction layer, which is the interface that the operating system provides to the application, is sometimes referred to as the *runtime environment*. The hardware and operating system of a machine are collectively called the *platform*. Above the operating system reside the applications which

are used to process and present data. The data itself and its representation can be considered a layer atop the application. Finally, many machines in today's systems are not isolated devices, and in fact, they are connected to other machines through a *network*. In general, five domains of MT techniques address dynamically changing the abovementioned software stack layers.

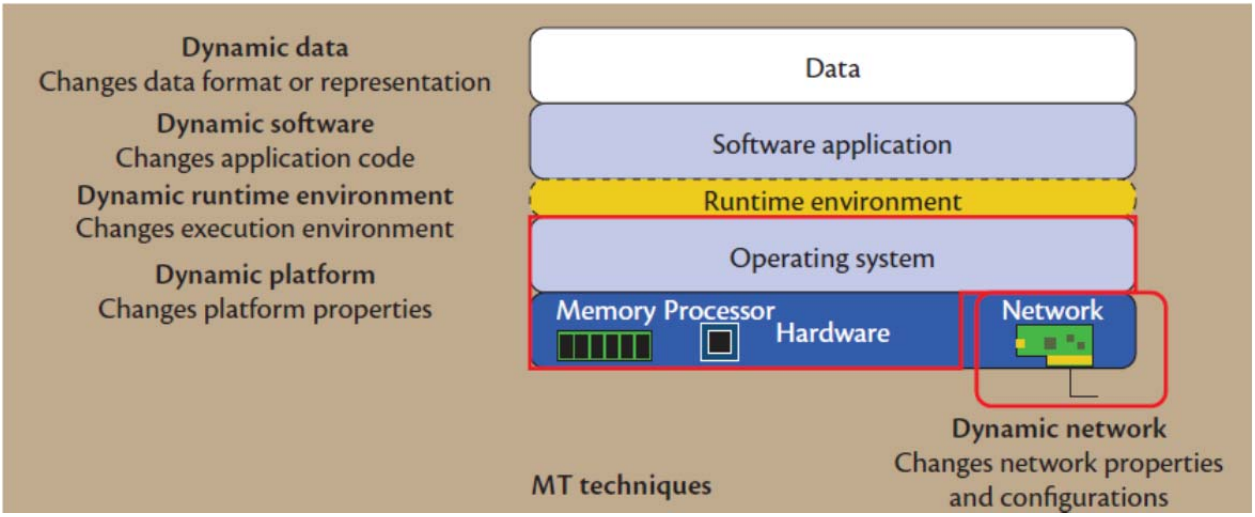

Figure 1: Different domains of cyber moving target techniques

Dynamic Network:

Techniques in the dynamic network domain change the properties of the network to complicate network-based attacks. One such technique frequently changes the Internet protocol (IP) addresses of the machines in an enterprise network [9]. This IP rotation technique can thwart rapidly propagating worms that use a fixed hit list of IP addresses to infect a network. Another technique, known as an overlay network, creates dynamically changing encrypted tunnels (i.e., encrypted communication connections over public networks).

Dynamic networks is an appealing class of techniques to reduce an adversary's ability to conduct reconnaissance on a network, map a defended network, or select specific hosts for a targeted attack. However, these techniques face two important obstacles to deployment.

First, because many dynamic network techniques lack a well-articulated threat model, it may be unclear to network defenders what threat needs to be mitigated and thus how best to deploy the defensive technique. Consider a technique that isolates a small group of machines from the larger network (or Internet). If hosts within the isolated network can still communicate to hosts beyond the isolated network, protected hosts may be vulnerable to any number of client-side attacks that exploit vulnerabilities within the unprotected hosts' web browsers or document viewers. For example, targeted spear phishing (fraudulent email messages that try to elicit information such as passwords to Internet accounts) could

penetrate a protected network through the network's connections to unprotected hosts. Dynamic network–based MT techniques do not address these types of attacks.

Second, many dynamic network techniques introduce randomization into the fundamental protocols that are used on the Internet. However, the effectiveness of this randomization at stopping attacks is unclear. Suppose an MT technique randomizes network identifiers (such as an IP address). If service discovery protocols such as the domain name service (DNS) are used to convert human-readable domain names to machine-readable IP addresses, these services may undo any potential security benefit obtained through the MT technique itself, provided that the attacker can issue DNS queries.

Dynamic Platform:

The dynamic platform domain consists of cyber defensive techniques that dynamically change the properties of the computing platform. Consider a system that runs a given application on top of multiple operating systems and hardware architectures. For example, the application can run on top of a platform consisting of the Fedora operating system and x86 processor architecture or a different platform consisting of the FreeBSD operating system and ARM processor architecture. Such a system can be implemented, for instance, by compiling the application to different processor architectures and implementing a platform-independent checkpointing mechanism to preserve the current state of the application during platform changes [4]. Such a system constitutes a dynamic platform moving target technique. Other examples of dynamic platform techniques include a voting system that runs an application on top of different platforms, each platform voting on the output of the system [5], or a system that randomizes the internals of the operating system that are unimportant for the correct functionality of the application.

The major benefit of the dynamic platform techniques is preventing platform-dependent attacks. Crafting a successful exploit against a system usually requires that an attacker consider the exact platform of that system. This is similar to the process of developing software for a given system. As a result, by changing the computing platform, an MT technique can mitigate attacks that are platform-dependent. An attacker can develop a stronger attack by incorporating different exploits against different platforms, but this will increase the cost and workload of the attack, which is the main goal of MT techniques. Note that dynamic platform techniques cannot mitigate attacks that target a higher-level application logic flaw and do not depend on the platform. For example, SQL injection attacks [cite], which are attacks that inject a malicious command into a database application using a flaw in the high-level logic of the application, are typically not mitigated by dynamic platform techniques.

While dynamic platforms MT techniques offer the potential to defeat platform-dependent attacks, they can increase the complexity of the overall system, are generally difficult to effectively manage, and can actually be detrimental to security if used inappropriately [6]. Perhaps the greatest challenge from a system complexity and management perspective is the synchronization of application state across the set of diverse platforms. Examples of such program state could include open data files, user input from a keyboard or mouse, or network traffic that needs to be correctly delivered to a specific running process (while correctly maintaining connection-specific state in the kernel). Synchronizing these resources among the dynamic platforms in real-time requires a complex management infrastructure that can migrate state with speed and agility. Reasoning about the correctness of this management infrastructure may be

challenging in practice, but at the very least, the necessity to keep program state synchronized across several distinct platforms increases system complexity considerably.

Another potential limitation of dynamic platforms is that requiring multiple distinct platforms can actually increase the attack surface of the system. Attack surface refers to components of the system that are exposed to a potential attacker and can be the target of the attack. Suppose that a dynamic platform MT technique migrates an application between three platforms: Linux, Windows, and Mac. If the attacker has an exploit that works on the Windows host, the attacker simply needs to wait until the application migrates to the Window host to launch the exploit and compromise the application. Making the program migration less predictable can help, provided that the attacker cannot reliably guess which platform is running the application. As a result, dynamic platform techniques are only effective in cases where the diversity is *in-series* and not *in-parallel*. In other words, the successful attack must require all platforms to be compromised, not any platform. For instance, if the attack requires a long time to succeed (long duration disruption of service), a dynamic platform can be helpful. Otherwise, for short-duration attacks, it can be detrimental to security.

Dynamic Runtime Environment:

Techniques in the dynamic runtime environment domain dynamically change or randomize the abstraction provided by the operating system to the applications, without hindering any important functions of the system. One of the most important abstractions in a computer system is memory. For various reasons including isolation of different applications, compatibility, and interoperability, memory locations presented to an application in most modern computer systems is not a direct representation of the actual physical memory. Rather, a redirection is applied by the operating system in an abstraction known as the virtual memory. A well-known dynamic runtime environment MT technique randomizes what addresses in the virtual memory are used by the application. The technique is typically referred to as Address Space Layout Randomization (ASLR) [7] and is implemented in most modern operating systems including Linux, Windows, Mac OSX, Android, and iOS. By randomizing the addresses, ASLR makes exploit development more difficult for an attacker because attackers do not know where to place their malicious code on the system. Other dynamic runtime environment techniques include those that change the processor instruction encoding (a.k.a. Instruction Set Randomization -- ISR), or finer-grained variants of ASLR in which smaller regions of memory are randomized.

Dynamic runtime environments are among the most practical and widely deployed MT techniques. Yet, despite their successes, there are two important weaknesses than can allow an attacker to circumvent the defense.

First, ASLR requires memory secrecy. That is, if the contents of memory are disclosed or leaked to an attacker, the attacker may be able to use this information to defeat ASLR. Such memory disclosures are possible via separate vulnerabilities (known as buffer over-read vulnerabilities), where the contents of memory are read beyond the allowed boundary, disclosing how memory has been randomized. Without strict memory secrecy, an attacker can still circumvent the protections provided by ASLR to launch code injection or code reuse attacks (e.g., ROP) [cite].

Second, the granularity of randomization in many ASLR implementations is low, which reduces the overall protection provided by the technique. For example in Linux, only the start location of certain memory regions is randomized by default. The executable program code itself is often not compiled with ASLR support. As such, this section of the program's memory is not protected and can be a vector of exploitation.

Dynamic Software:

Techniques in the dynamic software application domain (or simply the dynamic software domain) randomize or diversify the internals of the software application. One technique from this domain is called the multi-compiler [8], which creates different versions of software executables (binaries) from the same source code (e.g. written in C) that perform the same function, but are different internally. The different internals can arise from different actual processor instructions that are used during the compilation process or using the same instructions in different locations inside the executable. Note that a given copy of the executable with a given set of internals may never change, but various machines in an enterprise run different executables. In other words, this technique can create spatial diversity (diversity among many machines) as opposed to temporal diversity (diversity of one machine over time). The major benefit of dynamic software techniques is to mitigate the impact of large-scale attacks. If an exploit is crafted against a given variant of the executable, it will have a small chance of working against other variants of that executable. Hence, an attacker cannot compromise many machines at once. This is contrary to many existing systems where if an attacker develops malware, it can successfully compromise millions of machines running the same target application. In recent sophisticated attacks, attackers reuse parts of the benign code of the target application itself to achieve malicious behavior. Known as code reuse attacks or return-oriented programming (ROP) attacks [9], these attacks can successfully bypass existing defenses that detect or stop foreign pieces of code in order to mitigate attacks. Dynamic software techniques can be effective against such attacks by making the benign application code diverse.

Dynamic software techniques often use specialized compiler techniques to produce executable software variants with different and unpredictable memory layouts. These variants could use padding to make the size of memory regions unpredictable, or insert No-operation (NOP) instructions within executable code that do not perform any operation, but can make code reuse attacks harder to launch because they change the location of other instructions. These techniques, however, suffer from a variety of weaknesses.

First, recompilation to produce a software variant requires access to a program's source code, and is not compatible with proprietary, third-party software for which source code is not made available. Furthermore, reasoning about the correctness of the compiled variant can be challenging, since one cannot simply verify a cryptographic measurement of the executable file to ensure that the code has not been (maliciously) modified.

Second, software is often compiled with special optimization flags that reduce the space and/or computational complexity of the compiled binary code. MT techniques that explicitly compile the software to introduce randomness in the memory layout may not be compatible with space saving or

compute-time saving optimization passes performed by the compiler. Consequently, the dynamic software is unlikely to maintain the same performance properties as the ideally optimized compiled code.

Third, dynamic software techniques that use execution monitors to instrument and compare multiple versions of an executable introduce significant performance costs. For example, if an MT technique has two variants, there is at least a 2x performance cost relative to native execution of the application (in terms of processor, memory, and I/O utilization). This cost may be reasonable for protecting one or two applications where the highest degree of security is required, but likely does not scale to protect all applications running on a host.

Fourth, information leakage attacks can also be used against dynamic software techniques (similar to dynamic runtime environment techniques) to bypass them. If attackers can leak how an executable has been diversified, they can attack it as if it was not diversified at all.

Dynamic Data:

Techniques in the dynamic data domain change the format, syntax, representation, or encoding of the application data to make attacks more difficult. In this domain, the diversity can be temporal or spatial as well. One technique in this domain dynamically changes the representation of the user identifier (UID) in Linux operating systems. This identifier is used to determine what access rights a user has. One type of attack tries to increase the access level of a user in order to gain access to otherwise sensitive resources by changing the UID value to that of an administrator. This type of attack is an example of a larger class of attacks known as privilege escalation attacks. The UID randomization dynamic data technique can mitigate such an attack.

Dynamic data techniques offer the promise of protecting data from theft or unauthorized modification, but these techniques also suffer from two important weaknesses.

First, there can be a lack of diversity in the number of acceptable data encodings. For example, to encode binary data, one could use base-64 or hexadecimal which are both commonly used in practice, but there are few other accepted standards for data encoding. Additional non-standardized encodings are certainly possible, but may increase the complexity of interoperating with other system components.

Second, the use of additional data encodings may also increase the attack surface of the software. For each encoding type, the software must have the proper parsing code to encode and decode the data. The additional parsing code itself could have security-relevant software bugs.

Summary:

One way of understanding the benefits of MT techniques is by looking at the steps of a cyber attack that they are trying to mitigate. In order to successfully compromise a system, an attacker must progress through several steps, as depicted in Table 1. The first step is reconnaissance during which an attacker collects information about the target. The second step is accessing the victim during, which the attacker collects enough information about the configurations, applications, and software versions that are running on the target machine in order to develop an attack against it. During the third step, the attacker develops an exploit against a vulnerability in the target machine. Then the attack is launched in the next step which

may include, for example, a malicious network packet sent to the target machine or luring the user to click on a maliciously crafted link or using a malicious thumb drive. After the attack is launched and verified, the attackers may take additional steps to maintain their foothold on the target machine (i.e., persistence). Together these steps are referred to as the *cyber kill-chain*. Table 1 illustrates the main step of the cyber kill-chain that each domain of MT techniques tries to mitigate.

Table 1: Attack phases disrupted by each MT domain.

MT Domain	Attack Phases				
	Reconnaissance	Access	Attack Development	Attack Launch	Persistence
Dynamic Network	✓			✓	
Dynamic Platforms		✓	✓		✓
Dynamic Runtime Env.			✓	✓	
Dynamic Software			✓	✓	
Dynamic Data			✓	✓	

Effectiveness of Moving Target Techniques

Weaknesses of existing MT techniques motivated us to develop a set of criteria for evaluating their effectiveness. By studying attacks against well-known MT techniques we identified three major problems that contribute to the weaknesses of such techniques. First, in some cases the dynamic change in the system is too slow. In such cases, an attacker can observe the current state of the system using information leakage attacks, craft an attack against the current state, and compromised the system by launching the attack, all within the interval between two system changes. Second, in some other cases, the space of movement is too small. For example, consider a system that has two possible states. While attackers may not know the current state of the system, they will have 50% chance of success in attacking the system by pure guessing. In many MT techniques, attackers can also reduce the amount of uncertainty they are facing by quickly testing every possibility. This is also known as the *brute force* attack. Third, in some MT techniques, parts of an attack surface are dynamic, whereas other parts remain static. The static parts become a target of attack because they do not present any uncertainty for the attacker.

Using the above insights, we developed three criteria for evaluating an MT technique: *timeliness*, *unpredictability*, and *coverage*, as define below.

- **Timeliness:** The extent to which a movement can be applied between the time at which an attacker makes an observation and time at which an attack is completed.
- **Unpredictability:** The extent to which the outcome of current or future movements of the attack surface are indeterminable by an attacker.
- **Coverage:** The extent to which all elements of a defended attack surface are subject to movement.

Timeliness evaluates how fast the system moves. The actual time between movements depends on the attack model of concern for the technique. Hence, the definition of timeliness considers the possible attacker observations. In fact, an optimal MT technique should tie movement events to possible actions that can leak information to an attacker [cite].

Unpredictability evaluates the uncertainty faced by an attacker. A quantitative metric for unpredictability is entropy.

Coverage evaluates whether or not the MT technique moves every element of an attack surface. If some parts of the attack surface remain static, they can become the target of attacks.

We have also developed rigorous, quantitative metrics for these criteria [11], the discussion of which is beyond the scope of this book chapter. We use these metrics to evaluate MT techniques and analyze the protection they provide against cyber attacks.

Practical Considerations

When deciding to deploy an MT technique, there are many practical issues to consider. The defender should understand the potential performance impact of the MT technique on the system. Many MT techniques offer security against strong adversaries, but incur performance penalties, which could be prohibitively high depending on the application. Understanding the performance requirements of the system and the expected performance costs of the MT technique can help defenders make the right decision about deploying MT defenses.

Moreover, the defender should understand the effectiveness of the MT technique before it is deployed. Techniques that offer high effectiveness against realistic attack models should be selected before those that suffer from false positives or negatives, or those that protect against an unrealistic threat. Hence, an important part of this consideration is having a well-defined attack model that describes the exact types of attacks that are of concern and that are relevant to the system being protected.

Finally, the defender should understand the composibility of MT and non-MT techniques. MT techniques do not solve all security problems, but rather are best suited toward defending against specific threat models. For example, a defender may want to defend against code injection attacks using ASLR. But to achieve defense-in-depth, signature-based network monitoring can be used to examine network traffic in real time and drop all packets that appear to contain code injection payloads. Understanding how well MT and non-MT techniques can be composed to achieve the necessary protection is paramount to effective cyber resilience.

Future Directions

Future work in this area will focus on multiple directions. In designing new MT techniques or evaluating existing ones, it is imperative to analyze whether or not the additional complexity created by the randomization or diversification of the system's components is actually exposed to a potential attacker. As discussed earlier, many MT techniques create complexity in a system component, but when attacking the system, an attacker can avoid or bypass the complexity. This is usually achieved through information leakage attacks or attacks that work regardless of the specific internals of a component (e.g. higher level logic flaws in the application). The flip side of this challenge is to ensure that the complexity is not exposed to the system's operators and maintainers. Ease of deployment, operation, and maintenance are important for widespread deployment of cyber defensive techniques.

Furthermore, additional research is needed in the area of evaluation and assessment of MT techniques. For cyber security to transition from a craft to a science, it is important to have concrete, meaningful, and repeatable evaluation methods. An imperative part of evaluation is developing metrics that define measurement units of security and can be used to evaluate the absolute security offered by an MT technique and a comparative assessment of it against other techniques. Meaningful and objective evaluation of MT techniques can benefit from a variety of evaluation approaches including abstract analysis, modeling and simulation, testbed experimentation, and real-world measurements in operational systems.

Finally, an important future direction for MT research is to examine, study, and evaluate the composability of MT techniques with other MT and non-MT defenses. Cyber defenses in general and MT techniques specifically, do not provide a "silver bullet," protecting against every known cyber attack. As a result, in practice, multiple defenses should be combined to provide adequate protection. Understanding the impact of these defenses on each other, and the composability challenges that arise from them, is an open research area.

REFERENCES

1. D. Kaufman, An Analytical Framework for Cyber Security, Defense Advanced Research Projects Agency, 2011, available at www.dtic.mil/dtic/tr/fulltext/u2/a552026.pdf.

2. H. Okhravi, T. Hobson, D. Bigelow, and W. Streilein, "Finding Focus in the Blur of Moving Target Techniques," IEEE Security & Privacy, vol. 12, no. 2, 2014, pp.16–26.

3. H. Okhravi, A. Comella, E. Robinson, and J. Haines, "Creating a Cyber Moving Target for Critical Infrastructure Applications Using Platform Diversity," International Journal of Critical Infrastructure Protection, vol. 5, no. 1, 2012, pp. 30–39.

4. B. Salamat, A. Gal, T. Jackson, K. Manivannan, G. Wagner, and M. Franz, "Multi-variant Program Execution: Using Multi-core Systems to Defuse Buffer-Overflow Vulnerabilities," Proceedings of the IEEE International Conference on Complex, Intelligent and Software Intensive Systems, 2008, pp. 843–848.

5. H. Okhravi, J. Riordan, and K. Carter, "Quantitative Evaluation of Dynamic Platform Techniques as a Defensive Mechanism," pp. 405-425 in Research in Attacks, Intrusions and Defenses, Lecture Notes

in Computer Science, A. Stavrou, H. Bos, and G. Portokalidis eds. Cham, Switzerland: Springer International Publishing, 2014.

6. PaX Team, “PaX address space layout randomization (ASLR),” 2003, available at https://pax.grsecurity.net/docs/aslr.txt.

7. M. Franz, “E Unibus Pluram: Massive-Scale Software Diversity as a Defense Mechanism,” Proceedings of the 2010 Workshop on New Security Paradigms, 2010, pp. 7–16.

8. H. Shacham, “The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86),” Proceedings of the 14th ACM Conference on Computer and Communications Security, 2007, pp. 552–561.

9. S. Antonatos, P. Akritidis, E.P. Markatos, and K.G. Anagnostakis, “Defending Against Hitlist Worms Using Network Address Space Randomization,” Computer Networks: The International Journal of Computer and Telecommunications Networking, vol. 51, no. 12, 2007, pp. 3471–3490.

10. J. Seibert, H. Okhravi, and E. Söderström, “Information Leaks Without Memory Disclosures: Remote Side Channel Attacks on Diversified Code,” Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 54–65.

11. D. Bigelow, T. Hobson, R. Rudd, W. Streilein, and H.Okhravi, “Timely Rerandomization for Mitigating Memory Disclosures,” Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015, pp. 268–279.

This material is based upon work supported by the Department of Defense under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense.